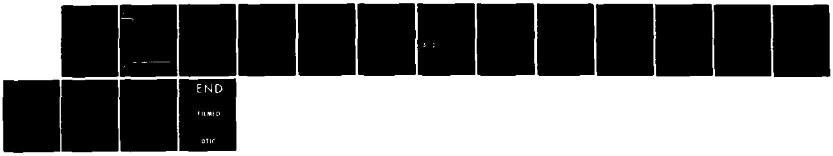
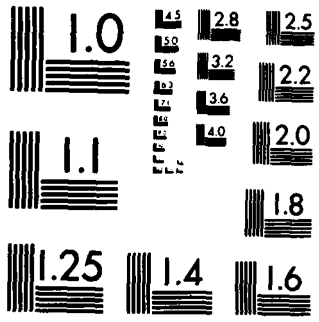

END
FILMED
DTIC


1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

AFHRL-TP-84-33

12

AIR FORCE HUMAN RESOURCES LABORATORY

AD-A150 342

# LOW ALTITUDE TEXTURE COMPARISON DATA BASE AND SMOOTH SHADED TEXTURE

By

Michael Phelps

OPERATIONS TRAINING DIVISION
Williams Air Force Base, Arizona 85240-6457

December 1984
Final Technical Paper for Period October 1982 - July 1983

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5000

85 01 30 009

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b RESTRICTIVE MARKINGS |
| 2a SECURITY CLASSIFICATION AUTHORITY | 3 DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited. |
| 2b DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S)<br>AFHRL-TP-84-33 | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a NAME OF PERFORMING ORGANIZATION<br>Operations Training Division — 6b OFFICE SYMBOL *(If applicable)*<br>AFHRL/OT | 7a NAME OF MONITORING ORGANIZATION |
| 6c ADDRESS *(City, State, and ZIP Code)*<br>Air Force Human Resources Laboratory<br>Williams Air Force Base, Arizona 85240-6457 | 7b ADDRESS *(City, State, and ZIP Code)* |
| 8a NAME OF FUNDING/SPONSORING ORGANIZATION<br>Air Force Human Resources Laboratory — 8b OFFICE SYMBOL *(If applicable)*<br>HQ AFHRL | 9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c ADDRESS *(City, State, and ZIP Code)*<br>Brooks Air Force Base, Texas 78235-5000 | 10 SOURCE OF FUNDING NUMBERS |

10 SOURCE OF FUNDING NUMBERS

| PROGRAM ELEMENT NO | PROJECT NO | TASK NO | WORK UNIT ACCESSION NO |
|---|---|---|---|
| 62205F | 1123 | 03 | 73 |

11 TITLE *(Include Security Classification)*

Low Altitude Texture Comparison Data Base and Smooth Shaded Texture

12 PERSONAL AUTHOR(S)

Phelps, Michael

| 13a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT *(Year, Month, Day)* | 15 PAGE COUNT |
|---|---|---|---|
| Final | FROM Oct 82 TO Jul 83 | December 1984 | 16 |

16 SUPPLEMENTARY NOTATION

17 COSATI CODES

| FIELD | GROUP | SUB-GROUP |
|---|---|---|
| 05 | 08 | |
| 05 | 09 | |

18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

image processing flight simulation
texturing

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Simulating precision low-level flight (at altitudes below 1,000 feet) is extremely difficult in current flight simulators with computer image generation (CIG). Their limited edge capacity precludes the creation of terrain that is as realistic and complex as the real world. This effort documents the techniques used to create smooth shaded texture on the digital image generator and to describe the layout of a data base constructed for the purpose of comparing its effectiveness to that of a plaid, checkerboard-type texture.

| | |
|---|---|
| 20 DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS | 21 ABSTRACT SECURITY CLASSIFICATION<br>Unclassified |
| 22a NAME OF RESPONSIBLE INDIVIDUAL<br>Nancy A. Perrigo<br>Chief, STINFO Office | 22b TELEPHONE *(Include Area Code)*<br>(512) 536-3877 — 22c OFFICE SYMBOL<br>AFHRL/TSR |

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

AFHRL Technical Paper 84-33 December 1984

# LOW ALTITUDE TEXTURE COMPARISON DATA BASE AND SMOOTH SHADED TEXTURE

By

Michael Phelps

OPERATIONS TRAINING DIVISION
Williams Air Force Base, Arizona 85240-6457

Reviewed and submitted for publication by

Robert B. Bunker
Chief, Technology Development Branch

## SUMMARY

Simulating precision low-level flight (at altitudes below 1,000 feet) is extremely difficult in current flight simulators with computer image generation (CIG). Their limited edge capacity precludes the creation of terrain that is as realistic and complex as the real world. The comparatively sparse and cartoon-like appearance of the scenery does not provide adequate visual cues that would enable the pilot to judge speed, altitude, distance from obstacles, or other factors necessary for a safe flight. To overcome these problems, various methods have been used to apply texture to the ground terrain. This texture most often resembles a checkerboard or plaid pattern of multicolored squares which spread along the path of flight. With such texturing, low-level flight becomes much easier.

Though checkerboard texture patterns make it possible to fly close to the ground, they add nothing to the realism of the scene and become monotonous after only a short time. The F-111 digital image generator (DIG) visual system (manufactured by Singer-Link) has a smooth shading algorithm which can be used to create terrain texture of a different sort. Smooth shading varies the intensity of the colors on the ground in a random fashion to produce texture that is more pleasing to the eye and more natural in appearance than regular predictable checkered patterns. It has not yet been determined whether this new texturing scheme is more effective in furnishing the needed visual cues for low-altitude flight.

The aim of this paper is to document the techniques used to create smooth shaded texture on the DIG and to describe the layout of a data base constructed for the purpose of comparing its effectiveness to that of a plaid checkerboard type of texture, referred to hereafter as gingham texture.

## PREFACE

This effort supports the Air Combat Training thrust of the Air Force Human Resources Laboratory (AFHRL). By describing an approach to provide the sensory and perceptual data base needed in the aircrew training device (ATD) design process, the effort directly supports the Engagement Simulation Technology subthrust. The work was accomplished in-house at the AFHRL Operations Training Division, Williams AFB. This paper consists of material that was written/submitted by the author and programmer and reviewed/reorganized at Williams AFB.

This effort was conceptualized, initiated, and managed by Robert B. Bunker, Chief of the Technology Development Branch at the AFHRL Operations Training Division. This effort documents the techniques used to create smooth shaded texture on the digital image generator and to describe the layout of a data base constructed for the purpose of comparing its effectiveness to that of a plaid, checkerboard-type texture.

# Low Altitude Texture Comparison Data Base and Smooth Shaded Texture

## Background and Rationale

Simulating precision low-level flight (at altitudes below 1,000 feet) is extremely difficult in current flight simulators with computer image generation (CIG). Their limited edge capacity precludes the creation of terrain that is as realistic and complex as the real world. The comparatively sparse and cartoon-like appearance of the scenery does not provide adequate visual cues that would enable the pilot to judge speed, altitude, distance from obstacles, or other factors necessary for a safe flight. To overcome these problems, various methods have been used to apply texture to the ground terrain. This texture most often resembles a checkerboard or plaid pattern of multicolored squares which spread along the path of flight. With such texturing, low-level flight becomes much easier.

Though checkerboard texture patterns make it possible to fly close to the ground, they add nothing to the realism of the scene and become monotonous after only a short time. The F-111 digital image generator (DIG) visual system (manufactured by Singer-Link) has a smooth shading algorithm which can be used to create terrain texture of a different sort. Smooth shading varies the intensity of the colors on the ground in a random fashion to produce texture that is more pleasing to the eye and more natural in appearance than regular predictable checkered patterns. It has not yet been determined whether this new texturing scheme is more effective in furnishing the needed visual cues for low-altitude flight.

The aim of this paper is to document the techniques used to create smooth shaded texture on the DIG and to describe the layout of a data base constructed for the purpose of comparing its effectiveness to that of a plaid checkerboard type of texture (referred to hereafter as gingham texture. Portions of the information contained here are from the Singer-Link Software Illustrator's Manual (see chapter 18 and pages 438-440).

## How Smooth Shading Works

Any object constructed on the DIG that has been flagged as three-dimensional (3-D) in the FDBGEN modeling program can be smooth shaded. Usually this is done in order to improve the stoogy image of an object built out of flat straight-edged polygons. A smooth-shaded object seems to be smoothly curved or rounded; the sharp edges between the faces are no longer noticeable.

The procedure is extremely simple and the results are dramatic. In FDBGEN, the modeler selects the Shading Home portion of the tablet menu and inputs the number of whatever is to be shaded. Individual or multiple clusters, objects, or faces can be shaded. Digitizing the button labeled COMPUTE VERTEX NORMALS completes the process.

The DIG determines the intensity assigned to a given face in the system as follows. An object such as a cube that uses the same color on all its faces is still clearly recognizable as a cube because each face is given a different intensity. The color remains constant but is displayed lighter or darker than every other face, depending on the orientation of the object. Without this feature, the cube would appear on the screen as a solid blob of color, and the only distinguishable angles would occur around the periphery. If the cube were displayed in front of or on top of a much larger surface of the same color, it would be virtually invisible.

As in the real world, the source of light in the visual simulator comes from a definite direction, referred to as the illumination vector. Each face of a 3-D model has a normal vector that is perpendicular to the plane of the face and pointing outward from the visible side of the face. In computing the intensity that will be given to the face, the system measures the angle between the illumination vector and the face normal vector. The greater the angle, the darker the intensity, because the face is considered to be on the opposite side of the object from the "sun", as the angle decreases, the intensity grows lighter.

The smooth shading routine creates normal vectors, which point outward from each individual vertex of a face. These vertex normals are not necessarily perpendicular to the face (or parallel to the face normal). Their direction is set automatically by the system when the cluster, object, or face is shaded, but can later be changed manually by the modeler. During the shading operation, the system averages the face normal vectors of adjoining faces and assigns that value to the vertex normals of the vertices that the faces share. From that point on, the face normal vectors are ignored in the intensity level computations. The measured angle is now between the vertex normal and the illumination vector, and the intensity that results from the calculations is assigned to the vertex and not to the face as a whole. This produces a face that varies in brightness from one edge to another. Since common vertices of adjoining faces also have common vertex normals, the shading will change smoothly from one face to the next, ignoring the face boundaries and preventing the display of "hard" edges.

## Creating Smooth Shaded Texture

As was mentioned before, a model must be declared as 3-D in order to be smooth shaded. This is done in Cluster Home when the cluster is first defined. The system then requires three numerical imputs (X, Y, and Z), to define every vertex of the model.

Smooth shaded texture consists of 3-D clusters that are square-shaped and perfectly flat. Each cluster is made up of square objects that are further subdivided into square faces. In the case of texture that lies on the ground plane, each vertex of the entire cluster has a common Z-coordinate value (i.e., o), and each face lies on the same plane as every other. The result is a two-dimensional model that has been flagged as 3-D.

After the grid of square faces, objects, and clusters has been modeled in FDBGN, it must be smooth shaded in Shading Home. This will create the vertex normal vectors extending from the corners of each square face. Since all the faces share the same plane and all the face normal vectors are parallel, the vertex normal vectors will also be parallel to each other and perpendicular to their faces. If FDBGN were canceled at this point and the data base file compiled and viewed in real time, the model would be a solid color without any shading whatever. The key to creating texture lies in manually resetting the vertex normal vectors after they have been created by smooth shading. This can be done in two ways: (a) by referencing each vertex, one at a time, in Shading Home and inserting the new values, or (b) by bringing faces into work by referencing each vertex and changing the vertex normal values in Vertex Home.

The vertex normals created by smooth shading are unit vectors with origins at their respective vertices. This means, in effect, that a vertex normal vector begins at (0, 0, 0) in its own individual coordinate system. Its length is exactly one unit of measure, and it may point in virtually any direction. The X, Y, and Z values associated with it (seen when the face is documented) represent the coordinates of the tip of the vector, using its own system once again.

Before the vertex normals are manually reset, each one will have identical coordinates: 0 in the X direction, 0 in the Y, and 1 in the Z. This occurs only because all the faces rest on the same plane, After resetting the vertex normals, the texturing effect will become apparent in real time. For all intents and purposes, the system's built-in "sunlight" has been fooled into treating the flat, shaded clusters as though they were rolling hills and valleys.

## Design Factors

There are many factors relating to data base design that must be considered when building smooth shaded texture. The following discussion will address the major issues and present reasons for the structure of smooth shaded texture data bases built up to this point.

It is recommended that shaded texture clusters, objects, and faces be square-shaped. This is not an absolute requirement but proves to be very convenient. During compilation of a data base file, the DIG must build a plane between every object of a cluster and between every cluster of the entire file. A poorly designed data base may contain objects and clusters which, by virtue of their shape or the way they are combined, cannot be separated from each other. In such a case, the compile will fail and the problem must somehow be rectified. Square objects or clusters that are all the same size and joined corner to corner will cause no problems when the separating plane tree is created. Square shapes can also cover an area without leaving gaps between faces, which would be difficult to achieve with other shapes arranged in separable combinations. In short, the square shape meets both essential requirements for smooth shaded texture: (a) it can be easily separated, and (b) it leaves no gaps in the texture.

The DIG can display only a limited number of edges at any one time. For this reason, careful consideration must be given to the amount of area to be covered with texture and the density of the texture itself. Smooth shaded texture uses huge numbers of edges and can easily over load the system.

The problem of edge overload is overcome in two ways. The first method involves the creation of null clusters with appropriate switching distances to cause the texture clusters to be replaced when they move far away from the viewpoint. In this manner only the texture that is in the general vicinity of the ownship will be displayed and the horizon, where most overload problems happen, will be kept relatively uncluttered. The switching distance at which cluster replacement occurs must be long enough to ensure that the texture will be ready in plenty of time as the viewpoint moves towards it, yet short enough to prevent the system from trying to process too much texture at once. Smooth shading greatly increases the number of calculations that must be done before a model can be displayed. A less-that-optimim switching distance will result in a noticeable lag in the time it takes for shaded texture clusters to appear. In an extreme case, this could be as long as several minutes. Unfortunately, there is no standard distance that will satisfy every situation. Each application of smooth shaded texture will be different, and only trial-and-error experimentation can provide the best results.

A second way to combat system edge overload is to control the density of the texture itself. If the square faces in each cluster are very small, the random texture patterns will be very dense. The effect is helpful in that it provides increased visual detail, but it also increases the number of displayed edges and shrinks the area the texture can cover at any given instant. By increasing the size of the faces, more area can be textured and displayed with fewer edges at the cost of decreased detail and visual impact. The designer must balance the two variables (texture density and texture area and arrive at an acceptable compromise.

The easiest way to cover a large area with smooth shaded texture is to model only one texture cluster and then copy it as many times as necessary to fill the space. The vertex normals on the edge of the cluster must be arranged in such a way that they will match the vertex normals along the edge of the adjoining cluster. In other words, faces that share the same points in space for their vertices must have identical vertex normal vectors at those vertices. If this is not the case, a "hard" edge will be observed on the display.

## Texture on Hills and Mountains

So far, mention has been made only of texture that follows the ground plane (Z=0). Smooth shading can also be applied to surfaces that are not parallel to the ground, such as hills or mountain ranges. This requires much more effort than flat texture, however. The design process can become extremely complicated, depending on the shapes and arrangement of the mountains; therefore, only general guidelines can be presented in this paper.

A typical mountain on the DIB consists of simple triangular faces put together in a "teepee" shape. To create a similar textured mountain, each face of the former must be subdivided into many smaller square faces. What represented only a single face on the old untextured mountain becomes an entire object with its own coplanar faces on the new. The mountain is no longer a single object but several objects, depending on the number of faces it had.

The first step must always be a drawing of the mountain produced on graph paper. Consider the paper as being the X, Y plane and draw the mountain in a top view looking straight down the Z-axis (similar to the orientation used by the system for cluster documenting). The squares on the graph paper will represent the small faces making up the texture. The scale of the drawing will, of course, determine the actual size of each square; hence, the density of the texture.

Each facet or side of the mountain will be built as an individual and separate object. If any one side has more than 35 to 40 faces, it must be further divided into two objects. This is determined on the drawing by numbering each sauare or part of a square that appears in the boundary of a side. Before going on, the objects must be checked for possible separability problems and redesigned if necessary.

It is easy to set the exact X and Y coordinates of every face from the graph paper drawing, but obtaining the Z coordinate for all the vertices that are above the ground plane poses a special problem. At times, the Z value will be obvious from the drawing, but either the X or the Y will not because the vertex does not lie on an intersection of grid lines. The equation $A=i(GX-x)+j(GY-y)+K(GZ-z)$ can be used to calculate the missing value of the three required coordinates. The variables i, j, and k represent the X, Y, and Z values of the normal to the plane of the mountain side; x, y & z represent the location of a known point on that plane; and the variables GX, GY, and GZ are the value of the vertex in question. Of these lists only two are known or deduced from the winner.

Obviously, a mountain of any size will have a great number of small texture faces. Caculating by hand the coordinates of each vertex of each face would take a tremendous amount of time. To speed up the process, a special

Fortran program was written for the DIG by Michael Arsenault of the Operations Training Division. Simply put, the program accepts from the user the inputs just mentioned and quickly calculates the missing value of GZ or GX or GY. It then loops back and solves the equation again for the next vertex.

To use the program it is necessary to find the normal to the plane that each side of the mountain creates. This normal is equal to the face normal vector that is perpendicular to all of the faces of that side. Perhaps the easiest way to determine the values of the face normal is to actually build the mountain in FDGEN (forgetting about the texturing) and get a face document of every side. Knowing the face normal values and having a list of coordinate values (two out of three for each vertex) obtained from the graph paper drawing, the designer is ready to use the program SHADE, FTN to compute the missing numbers. The output of the program is a face-by-face, object-by-object listing of vertex coordinates, which is all that is needed to build the mountain.

When the mountain has been constructed, it must be smooth-shaded and then each vertex normal vector must be manually redirected. Once again, the graph paper drawing can be extremely helpful. Small arrows (representing the vertex normals) drawn from the face vertices can give a clue as to the "randomness" of the texture and help the modeler make sure that each shared vertex has the same vertex normal.

This concludes a discussion on the theory and general technique of smooth shaded texturing. The following is a more specific description of the Low Altitude Texture Comparison data base (LATCOM, RTB) developed by the author and of the problems encountered during its development.

## Low Altitude Texturing Comparision Data Base

The Low Altitude Texturing Comparison Data Base (LATCOM.RTB) was completed in March 1983 on the Singer-Link DIG visual system at the AFHRL Operations Training Division. It was designed to assist in the evaluation of two relatively new texturing techniques that make low-level flight in a visual simulator a little more practical. This paper briefly describes the layout of the data base, problems encountered in its construction, and what documentation is available.

The LATCOM.RTB data base provides six different test conditions meant to be flown at altitudes of 100 to 300 feet. Each separate condition consists of a right and a left turn segment in the form of a simplified valley 3,000 feet wide, 8,000 feet long, and bordered on either side by a chain of mountains approximately 1,000 feet high. Some of the turn segments have 100-to-200 foot-high ridges extending across the valley floor. Detailed maps of each segment of every test condition are contained in the documentation.

Three of the test conditions display gingham texturing (a plaid pattern using a special hardware feature of the DIG to change the colors of overlapping faces), and the remaining three utilize the smooth shaded texturing described in the first section of this paper. Each of the two texture types has a test condition without ridges, one with plain ridges, and one with textured ridges.

All the data base and library files used to create the LATCOM.RTB data base have been backed up on one of five magnetic tapes being stored in the DIG modeling room. The following is a listing of the file names on each tape:

Tape #1: Untranroed .DBF files

| | |
|---|---|
| SSTF23.DBF | SSTF34.DBF |
| SSTM23.DBF | SSTM34.DBF |
| SSTR23.DBF | SSTR34.DBF |
| LGTF5.DBF | LGTF10.DBF |
| LGTM5.DBF | LGTM10.DBF |
| LGTR5.DBF | LGTR10.DBF |

Tape #2: Untranroed .RTB files and library files:

| | |
|---|---|
| SSTF23.RTB | SSTF34.RTB |
| SSTM23.RTB | SSTM34.RTB |
| SSTR23.RTB | SSTR34.RTB |
| LGTF5.RTB | LGTF10.RTB |
| LGTM5.RTB | LGTM10.RTB |
| LGTR5.RTB | LGTR10.RTB |
| RIDGE.DBF | MIKE.DBF |

Tape #3 Tranroed .DBF files:

| | |
|---|---|
| STF23.DBF | STF34.DBF |
| STM23.DBF | STM34.DBF |
| STR23.DBF | STR34.DBF |
| GTF5.DBF | GTF10.DBF |
| GTM5.DBF | GTM10.DBF |
| GTR5.DBF | GTR10.DBF |

Tape #4 Tranroed .RTB files:

| | |
|---|---|
| STF23.RTB | STF34.RTB |
| STM23.RTB | STM34.RTB |
| STR23.RTB | STR34.RTB |
| GTF5.RTB | GIF10.RTB |
| GTM5.RTB | GTM10.RTB |
| GTR5.RTB | GTR10.RTB |
| PATHS.DBF | PATHS.RTB |

An explanation of naming conventions for the files may be found on the master data base map contained in the documentation. In addition to the maps and drawings, the documentation of this data base contains listings of the vertex coordinates; vertex normals; face, object, and cluster numbers; edge counts; hardcopy documents; and many more items collected during the modeling and design phase.

Brief mention has already been made of certain types of problems that arise during construction of smooth shaded texture, such as edge overloading and selecting the best switching distance. Other difficulties relate to specific applications and will not occur with every data base design. Hindsight is always much better than foresight, and it would be possible now to build a similar data base in a quarter of the time. Much time was wasted discovering through trial and error what would work and what would not.

The LATCOM.RTB data base borrows its mountainous turn sections from the LADDER data base. LADDER (for Low Altitude Data base Development, Evaluation and Research) was assembled by Camille Thomas of Singer-Link. The LADDER turn sections already answered questions relating to the size of the area blocks that make up the total gaming area. The blocks are 9,000 feet per side. Since the texture clusters were 2,000 feet on a side, they did not cover the entire block without overlapping. Every cluster that overlapped the area block boundaries had to have its excess deleted before compiling. The cluster hulls built around a supercluster and its subclusters also had to hold to the strict boundaries.

In this data base, the superclusters assigned to the textured clusters are called "null" clusters. This is actually a misnomer as they really do contain something. Each one is a point with a specific color and location. When the switching distance is reached, the display changes from texture to this single point or vice versa. It was necessary to create this type of supercluster only because the mountains switch in at 30,000 feet and thus cannot share the same "null" cluster as the texture. If the mountains and the texture were made to switch at the same distance, only one true null cluster would have been necessary for each area block. There can be no more than one truly "null" (empty) cluster per area block, but there can be many superclusters that are single points. All the subclusters of a given supercluster must have the same switching distance. Every bit of the smooth shaded texture should be assigned a supercluster. A large amount of it (more than 15 or 20 clusters of the type used in LATCOM.RTB) without texture clusters would give rise to an interesting phenomenon. A compile would be successful according to the documentation on the console terminal or printer, but nothing at all would be displayed in real time. This was only correctable when the texture was split up into smaller area blocks and assigned superclusters.

Most of the problems encountered in data base generator program FDBGEN reside in Maneuver Home. The Copy/Translate and Put fundtions all caused problems at one time or another. Repeating the function over again was the only solution.

It was discovered during assembly of the LADDER data base that the compiler sometimes stumbles over construction vertices in the data base. Most of these vertices were deleted from LADDER, but the turn sections borrowed for LATCOM still contain a few (such as cluster 13 objects 8 face 2 in the files STF34, STM34, and STR24). They revealed themselves during compiles of the tranroed sections and therefore had to be dealt with.

The TRANRO utility program had a strange problem at one point. It claimed that all the objects and faces of a particular cluster had been successfully moved but that the cluster itself had not budged from its original location. This seemingly impossible situation was corrected by bringing the tranroed .DBF file up in FDGEN, and the "Referencing" and "Object Completing" each object one at a time.

Aside from the effort needed in finding the coordinates, the texturing of the ridge clusters posed no problems. There is a problem setting them to respond correctly to a crash with the ownship, and the solution has yet to be worked out. Textured ridges in the latest revision of the data base have extra faces built inside so that each object is an enclosed space rather than merely a plane in space. This has given a correct crash indication in all possible orientations but has caused other odd problems. First, after the new hidden faces were added, the time required to compile the .DBF files SSTR23 and SSTR34 jumped from 1 to 3.5 hours. Second, the dispaly now shows an irritating flashing along the object boundaries of the textured ridges. The effect is observable only from out-of-the-ordinary viewpoints; a pilot flying throught the test conditions at a low altitude would be unlikely to notice it. More experimentation is necessary to solve this problem.